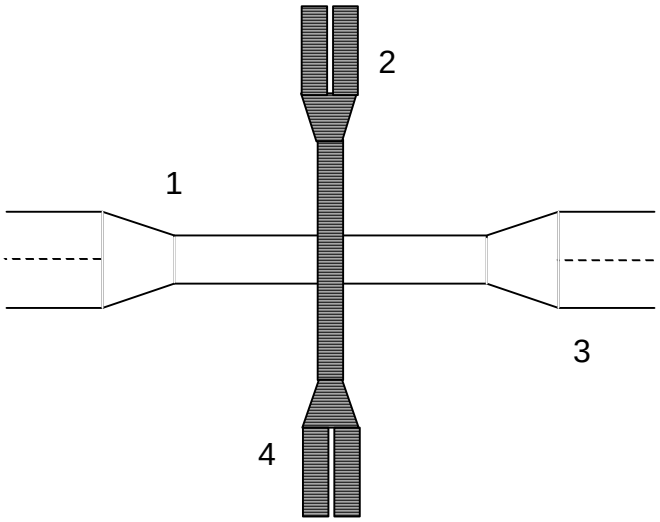
PRINCIPI DI SISTEMI OPERATIVI

ESERCIZIO del 15 LUGLIO 2005

Un **passaggio a livello** regola l'incrocio tra una **strada** e una **ferrovia**, e viene attraversato da due tipi di **veicoli**: **auto** e da **treni**. Esso è stretto, e permette il passaggio delle auto a senso unico alternato (direzioni 1 e 3), e dei treni su un binario unico (direzioni 2 e 4). Naturalmente in ogni istante possono passare solo auto o treni, ed in una sola direzione. I treni hanno la precedenza sulle auto, per cui le auto non possono passare quando il treno è in attesa che si liberi il passaggio a livello.

Si implementi una soluzione usando il costrutto monitor per modellare il **passaggio a livello** e i processi per modellare le **auto** e i **treni** e si descriva la sincronizzazione tra i processi. Nella soluzione si massimizzi l'utilizzo delle risorse. Si discuta se la soluzione proposta può presentare starvation e in caso positivo per quali processi, e si propongano modifiche e/o aggiunte per evitare starvation.



```
program passaggio a livello

type      tipo = (auto, treno);  { tipo di veicolo }
          dir = (1, 2, 3, 4);  { direzione: 1 e 3 auto, 2 e 4 treni}

type veicolo = process(t: tipo, d: dir)
begin
      p.entra(t, d);
      <attraversa il passaggio a livello>
      p.esci;
end


type passaggio = monitor

{ variabili del monitor }
var  nveicoli : integer;
      { veicoli che impegnano il passaggio a livello }
      cur_dir : dir;
      { direzione corrente }
      coda : array [dir] of condition;
      { code su cui sospendere i veicoli }

procedure entry entra (t: tipo, d: dir)
begin
      if (d <> cur_dir and nveicoli <> 0) or
      { se il passaggio è impegnato in altre direzioni }
        (t = auto and (coda[2].queue or coda[4].queue)) then
      { o se il veicolo è un'auto e c'è il treno in attesa }
            coda[d].wait;

      { occupa il passaggio }
      cur_dir = d;
      nveicoli++;
end
```

```
procedure entry esci
begin
        { libera il passaggio }
        veicoli--;
        if (nveicoli = 0)
        { se il passaggio è libero sveglia una delle altre direzioni
dando la precedenza ai treni }
                if (coda[2].queue) then
                        while (coda[2].queue) do
                                coda[2].signal;
                else if (coda[4].queue) then
                        while (coda[4].queue) do
                                coda[4].signal;
                else if (coda[1].queue) then
                        while (coda[1].queue) do
                                coda[1].signal;
                else if (coda[3].queue) then
                        while (coda[3].queue) do
                                coda[3].signal;

begin { inizializzazione delle variabili }
        nveicoli := 0;
        cur_dir := 1;
end

var   p: passaggio; { il nostro monitor }
        v1, v2, ... : veicoli (auto|treno, 1|2|3|4);

begin end.
```

Starvation

La soluzione presenta due tipi di starvation. La prima si verifica se i treni continuano a passare ritardando indefinitamente le auto. La seconda si verifica se i veicoli (treni o auto) in una determinata direzione ritardano indefinitamente i veicoli nella direzione opposta.

Si possono risolvere entrambi imponendo che dopo un certo numero di passaggi consecutivi in una direzione venga data la precedenza ai veicoli in un'altra direzione, ad esempio ruotando la priorità tra le 4 direzioni.

Inoltre, nella soluzione proposta vengono svegliati sempre prima i treni in direzione 2 e le auto in direzione 1; una politica più corretta dovrebbe prevedere una rotazione delle direzioni considerate per il risveglio.

Nota

Il tipo può essere ricavato dalla direzione; per chiarezza è stato comunque usato un parametro.